Test Plan/Design/Analysis

Alex Cone’ (aec2975)

Karime Saad (ks38728)

Test Plan

*Black Box Testing.

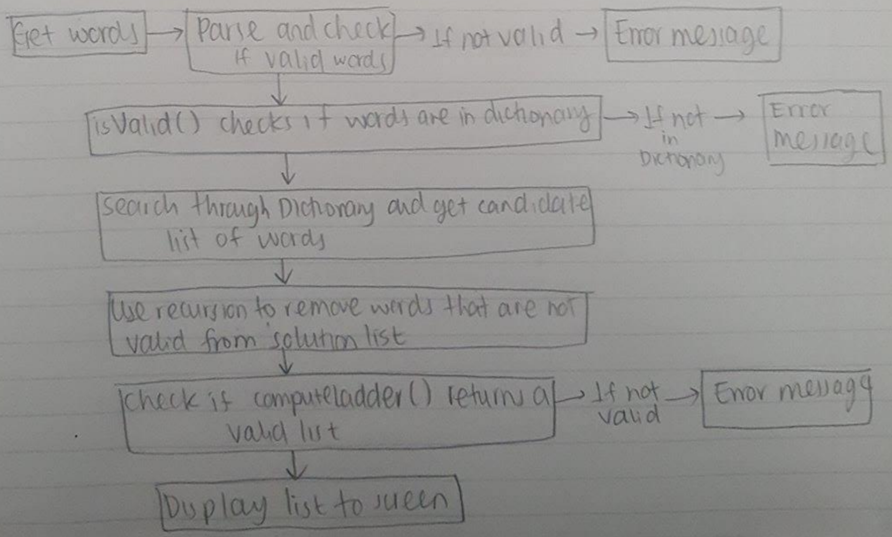
- Input: Two 5 letter words with one space in between that we are sure will output a word ladder.
Desired Output: A valid word ladder

- Input: Two words with one or both words having less than 5 letters.
Desired output: Error message - Invalid Output

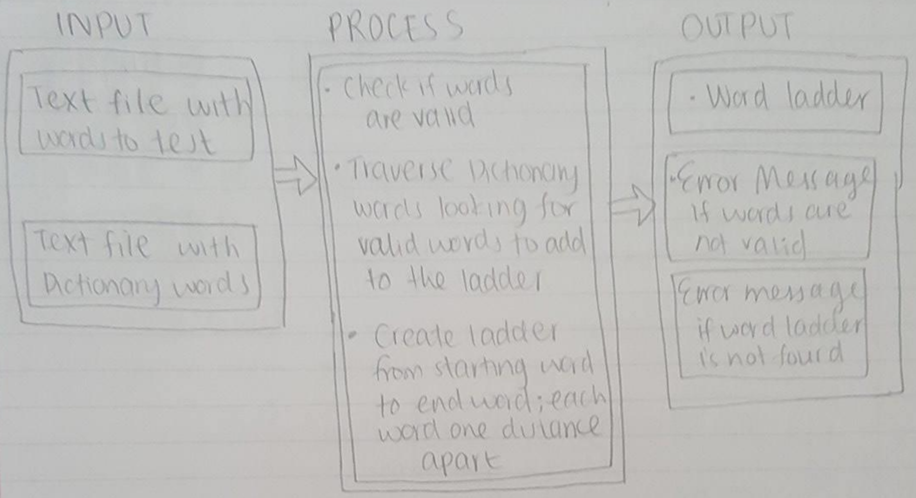
- Input: Two 5-letter words not in the dictionary
Desired Output: Error message - Words not in the dictionary

- Input: Two 5-letter words that will not have a word ladder
Desired Output: Error message - There is no word ladder between the two words.

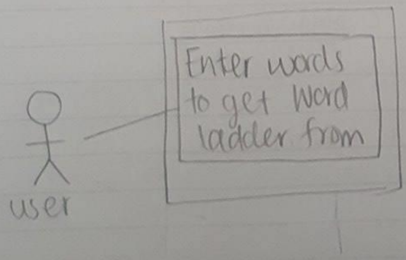
* White Box Testing.


Get words
Parse and check if valid words
If not valid
Error message
isValid() checks if words are in dictionary
If not in Dictionary
Error message
Search through Dictionary and get candidate list of words
Use recursion to remove words that are not valid from solution list
Check if computeladder() returns a valid list
If not valid
Error message
Display list to screen

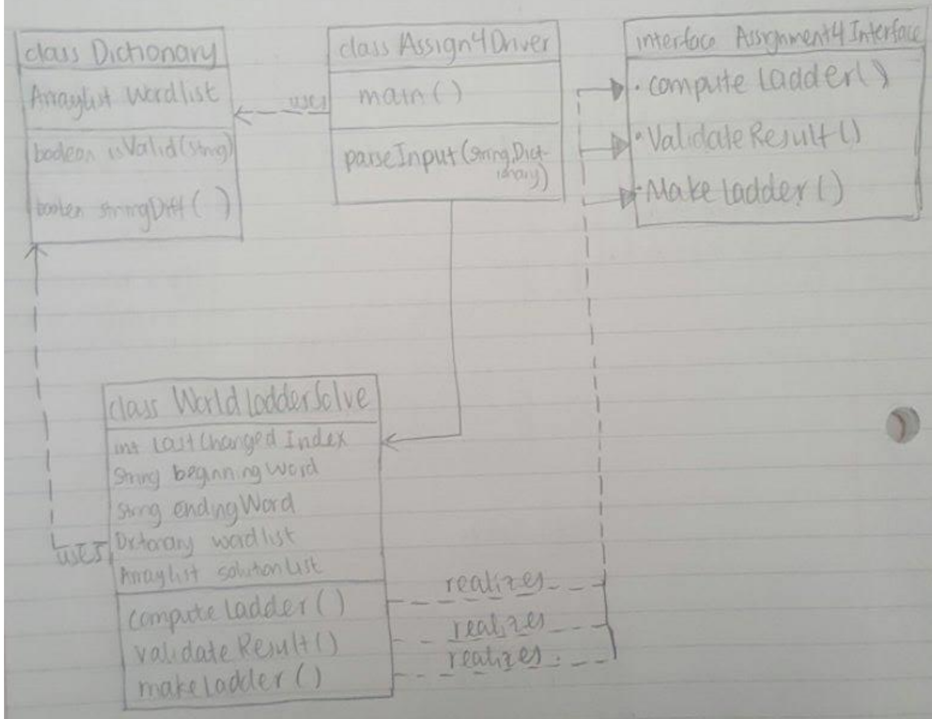
IPO diagram


INPUT
PROCESS
OUTPUT
Text file with words to test
Text file with Dictionary words
• Check if words are valid
• Traverse Dictionary words looking for valid words to add to the ladder
• Create ladder from starting word to end word; each word one distance apart
• Word ladder
• Error Message if words are not valid
Error message if word ladder is not found

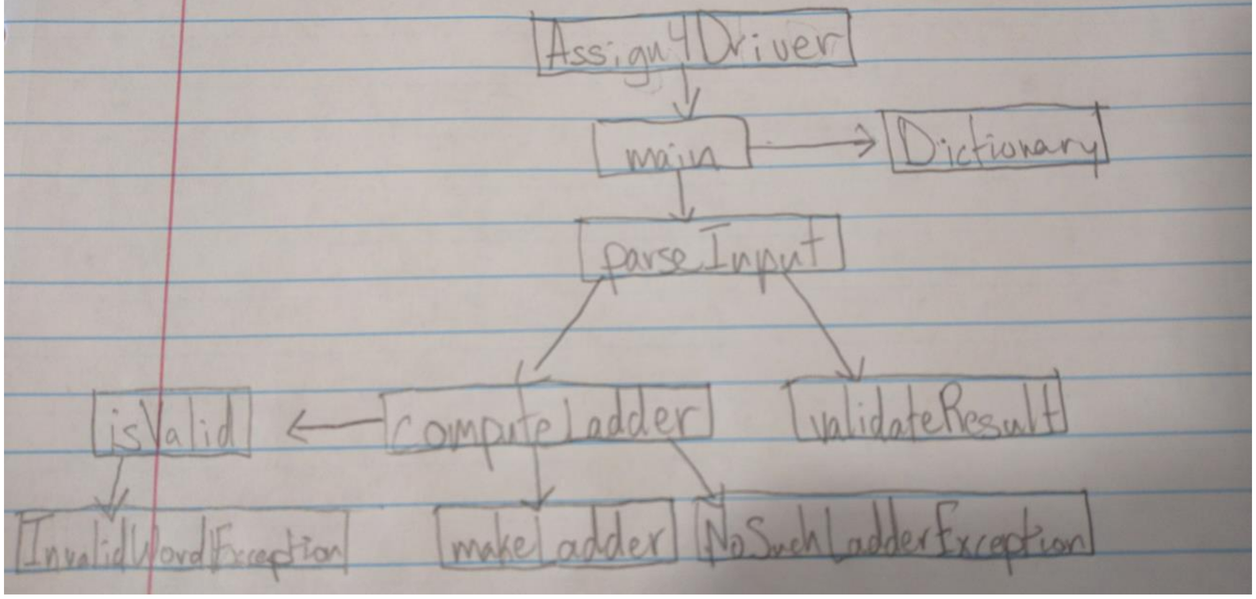
USE case

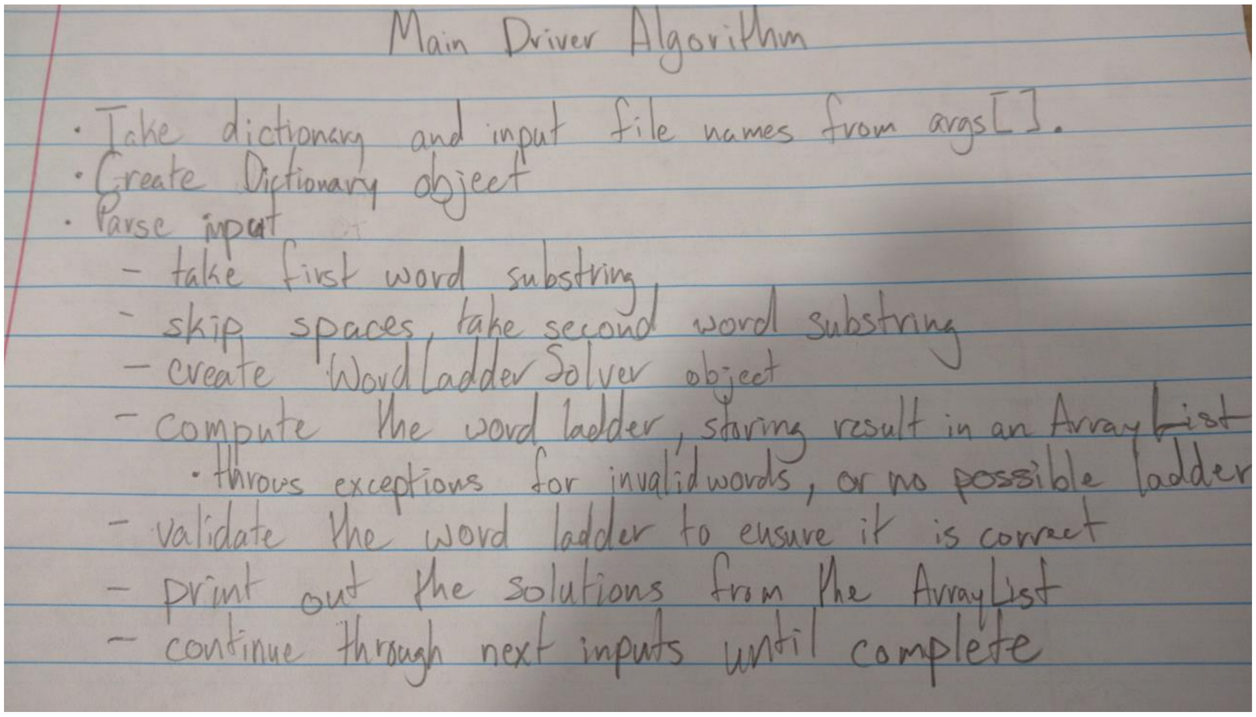


UML model


class Dictionary
Arraylist Wordlist
boolean isValid(String)
boolean StringDiff( )
uses
class Assign4Driver
main( )
parseInput(String,Dictionary)
interface Assignment4 Interface
· compute Ladder( )
· Validate Result( )
· Make ladder( )
class World ladderSolve
int Last Changed Index
String beginning Word
String endingWord
Dictionary wordlist
Arraylist solution List
compute Ladder( )
validate Result( )
makeLadder( )
uses
realizes
realizes
realizes

Functional Block Diagram





Main Driver Algorithm

- Take dictionary and input file names from args[].
- Create Dictionary object
- Parse input
  - take first word substring
  - skip spaces, take second word substring
  - create WordLadderSolver object
  - compute the word ladder, storing result in an ArrayList
    - throws exceptions for invalid words, or no possible ladder
  - validate the word ladder to ensure it is correct
  - print out the solutions from the ArrayList
  - continue through next inputs until complete

Rationale

Our design follows how real world objects work together. The word ladder solver, which in the real world would be a person, can access the dictionary in order to look through it to find possible words. This person can use the dictionary to check if the words are valid, just like how our Dictionary object can check if a word is within it or not. We chose to use DFS with recursion to solve this problem due to ease of coding. After using this method though, we have found that using BFS can lead to a more efficient program as well as finding shorter word ladders. Both methods can have added difficulty though due to debugging nested loops and recursion. As a user who may not care of the length of the word ladder DFS works fine and gets the job done. Our design is fairly flexible by allowing the use of the Dictionary and WordLadderSolver classes in other ways. The Dictionary class especially could be used for many other programs involving Strings. Lastly, our design adheres well to principles of good design. Our main driver works directly with the WordLadderSolver which in turn works directly with the Dictionary. Data that is part of these objects are hidden unless needed to be used so that the user does not have access to info they do not need.